EV205822882

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

RESOLVING A DISTRIBUTED TOPOLOGY TO STREAM DATA

Inventors:

Kirt A. Debique

Thomas A. Thornton

Troy D. Batterberry

Nadim Y. Abdo

Alexandre A. Grigorovitch

Dale A. Sather

Roland Y. Ayala

Eduardo P. Oliveira

ATTORNEY'S DOCKET NO. **MS1-1538US**

RESOLVING A DISTRIBUTED TOPOLOGY TO STREAM DATA

TECHNICAL FIELD

[0001] The present invention generally relates to streaming data, and more particularly related to resolving a distributed topology to stream data.

BACKGROUND

[0002] Streaming data provides increased functionality to a user such that the user may quickly receive the data. Without streaming, if the entire amount of the data was needed to be received from a source device before it was output by a client device, the user may experience a delay in rendering the data at the client device. By streaming the data, the delay encountered by the user may be lessened. Data streaming may be used to provide "real-time" rendering of data.

[0003] To stream data, data is transmitted from a source device in a streaming or continuous fashion, generally using data packets, for rendering at a client device as the data arrives, as opposed to data that is not rendered until an entire file which includes the data is available at the client device. Streaming may be used for a variety of types of data, such as video data, audio data, media data, and the like. A stream of video data provides a sequence of "moving images" that are transmitted and displayed when the images arrive. Likewise, a stream of audio data provides sound data that is played as the audio data arrives. A stream of media data includes both audio and video data.

[0004] Previously, use of peripheral devices which involved the use of streams of data was accomplished in a local manner. For example, peripheral devices, such as source peripheral devices (such as cameras, microphones, and capture cards) and rendering devices (such as display monitors and speakers) were physically attached to a single

computing device so that data streamed by the source peripheral device could be rendered by the rendering device. Rendering the data may include transforming the data into a form which is suitable to be output in a tangible form by a rendering device, such as displayed on a display device, played by an audio speaker, and so on.

[0005] If remote access to the streams of data was desired, the remote access was limited by the local manner in which the data was provided. For example, to provide remote access to data which was streamed by the source peripheral device, the data was output, compressed, decompressed, rendered, captured, and recompressed before it was streamed to a client device. However, when capturing and recompressing the data before streaming to the client device, portions of the data as output by the source peripheral device may be lost. Therefore, the data which was streamed to the client device was degraded. This reduced the experience of the user when viewing and/or listening to the data, such as by experiencing choppy movement in a display of video data and incomplete playback of audio data. Additionally, capturing and recompressing the data by the source device is resource intensive, thereby limiting such functionality to devices having sufficient processor and data storage resources. Further, capturing and recompressing the rendered data may slow the streaming of the data to the client device, thereby limiting the "real-time" nature of the data. Moreover, if more than one destination was desired for streams of data, the amount of resources which were used to support multiple streams was further increased.

[0006] Accordingly, there is a continuing need to improve the streaming of data.

SUMMARY

[0007] A distributed media session is described which, when executed, resolves a distributed topology that references software components for streaming data from a source device over a network to a client device. The distributed topology may be thought of as a framework or plan of software components that will be used to stream data from a source device over a network to a client device. Thus, when the software components of the distributed topology are executed, data is streamed from the source device to the client device over the network. A distributed software infrastructure of these software components may be built based on the distributed topology through execution of the distributed media session. Execution of the software components of the distributed software infrastructure allows data to be streamed from the source device over the network to the client device.

[0008] The distributed media session, when executed, may provide a federated (i.e. centralized) mechanism for resolving the distributed topology. For example, when executed, the distributed media session may take into account the capabilities of both the source and client devices when resolving the distributed topology. Additionally, the distributed media session, when executed, provides a federated mechanism for resolving the distributed topology such that the distributed media session may be executed by the client device, the source device, and/or a third party device. The executing of the distributed media session may also provide a federated mechanism for control. The software components, for instance, on both the source and client devices may be controlled by a single entity that represents such a federated mechanism.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] FIG. 1 is an illustration showing an exemplary implementation of a system that provides for streaming data that includes a parent distributed media session (DMS) for streaming data from a source device to a client device.

[0010] FIG. 2 is an illustration showing an exemplary implementation in which a computing environment utilizes the parent DMS of FIG. 1 to stream data.

[0011] FIG. 3 is an illustration showing an exemplary implementation of parent-child relationships of the parent DMS of FIG. 1 with a child DMS.

[0012] FIG. 4 is an illustration of an exemplary implementation which depicts the parent DMS that provides remote access functionality using a federated approach by employing the child DMS of FIG. 3.

[0013] FIG. 5 is an illustration of an exemplary procedure in which the parent DMS of FIG. 1 builds a distributed software infrastructure for streaming data from a source device to a client device.

[0014] FIG. 6 is an illustration of an exemplary implementation of resolving a remote source distributed topology employing the procedure of FIG. 5.

[0015] FIG. 7 is an illustration of the exemplary implementation of a remote source distributed topology that is further resolved from the remote source distributed topology of FIG. 6.

[0016] FIG. 8 is an illustration of the exemplary implementation of a remote source distributed topology that is fully resolved from the remote source distributed topology of FIG. 7.

[0017] FIG. 9 is an illustration of a hierarchy of control in the exemplary implementation shown in FIG. 8.

[0018] FIG. 10 is an illustration of an exemplary resolved remote sink distributed topology that is resolved through the procedure of FIG 5.

[0019] FIG. 11 is an illustration of a hierarchy of control in the exemplary implementation shown in FIG. 10.

[0020] FIG. 12 is an illustration of exemplary implementation of a proxy being executed on the client device of FIG 1.

[0021] FIG. 13 is an illustration of exemplary implementation of a proxy being executed on the source device of FIG 1.

[0022] FIG. 14 is an illustration showing an exemplary implementation of the proxy of FIG. 12 configured as a source proxy.

[0023] FIG. 15 is an illustration showing an exemplary implementation of the proxy of FIG. 13 configured as a sink proxy.

[0024] FIG. 16 is an illustration showing an exemplary implementation of a remote source distributed topology that includes the source proxy of FIG. 14.

[0025] FIG. 17 is an illustration of an exemplary implementation that depicts a hierarchy of control in the remote source distributed topology as shown in FIG. 16.

[0026] FIG. 18 is an illustration showing an exemplary implementation of a remote sink distributed topology that includes the sink proxy of FIG. 15.

[0027] FIG. 19 is an illustration of an exemplary implementation which depicts a hierarchy of control in the remote sink distributed topology as shown in FIG. 18.

[0028] FIG. 20 is an illustration of an exemplary implementation in which a system is shown that includes a third party device that is communicatively coupled to source and client devices over the network of FIG. 1.

[0029] FIG. 21 is an illustration showing an exemplary implementation of a third party distributed topology implemented in the system shown in FIG. 20.

[0030] FIG. 22 is an illustration showing an exemplary implementation of a hierarchy of control in the implementation depicted in FIG. 21.

[0031] FIG. 23 is an illustration showing an exemplary implementation of a plurality of source devices that stream data to the client device of FIG. 1.

[0032] FIG. 24 is an illustration showing an exemplary implementation of a distributed topology resolved for the system of FIG. 23.

[0033] FIG. 25 is an illustration showing an exemplary implementation of a system in which execution of the distributed media session resolves the distributed topology, using the procedure of FIG. 5, in an optimized manner based on the capabilities of source and client devices to interact with streaming data.

[0034] FIG. 26 is a flow diagram depicting an exemplary procedure in which a distributed software infrastructure of software components is built and data is streamed from a source device to a client device for rendering by the client device.

[0035] FIG. 27 is an illustration of an exemplary operating environment.

[0036] The same numbers are used throughout the disclosure and figures to reference like components and features.

DETAILED DESCRIPTION

[0037] **Overview**

A distributed media session (DMS) is described which, when executed, resolves a distributed topology that references software components for streaming data from a source device over a network to a client device. The distributed topology may be thought of as a software framework or plan for streaming data from the source device to the client device over the network using the referenced software components. When executed, the DMS may resolve the distributed topology by discovering the capabilities of the source device to stream data that is to be rendered and by discovering the capabilities of the client device to render the stream of data from the source device. From these two discoveries, the DMS builds a distributed topology that references software components that are suitable to stream data from the source device to the client device over the network. The distributed topology may then be utilized by the DMS to build a distributed software infrastructure that, when executed, allows data to be streamed from the source device over the network to the client device.

[0038] The DMS, when executed, may provide a federated, i.e. centralized, mechanism for resolving the distributed topology. For example, when executed, the DMS may take into account the capabilities of both the source and client devices when resolving the distributed topology. Additionally, the DMS provides a federated mechanism for resolving the distributed topology such that the DMS may be executed by the client device, the source device, and/or a third party device. The DMS may also provide a federated mechanism for control such that software components on both the source and client devices may be controlled by a single entity, such as by an application interacting with the DMS and/or the DMS itself.

[0039] **Distributed Media Session (DMS)**

FIG. 1 is an illustration of an exemplary implementation showing a system 100 that streams data. The system 100 includes a source device 102 and a client device 104 which are communicatively coupled over a network 106. The network 106 may include any of a local area network (LAN), a wide area network (WAN), and the Internet.

[0040] The source device 102 provides a stream of data to be rendered. The source device 102 may include a source peripheral device 108 which outputs data. The source peripheral device 108 may be configured as a camera 108a, microphone 108b, capture card, and the like. The source device 102 may also include a computing device 110 which is communicatively coupled to the source peripheral device 108 over a local connection, such as by using a universal serial bus (USB) port, IEEE 1394 port, parallel port, and so on. The computing device 110 includes a processor 112 for executing one or more software components and a memory 114 for storing the software components and data. The source device 102 may also include a rendering device 116 for providing a tangible output of the data to a user interacting with the rendering device 116. The rendering device 116 may be configured in a variety of ways, such as a display device 116a, a speaker 116b, and the like. The rendering device 116 is also communicatively coupled to the computing device 110 using a local connection.

[0041] Although the source device 102 illustrated in FIG. 1 is configured to include a source peripheral device 108 which is communicatively coupled to a computing device 110, the source device 102 may be configured in a variety of ways. For example, the source device 102 may be configured as a network-ready device which does not include a separate computing device 110 to connect to the network 106, such as by including a

processor, memory, and network hardware components itself. In another example, the source device 102 may be configured as a media server that provides a stream via a streaming protocol that is provided through execution of software by the source device 102.

[0042] The client device 104 may also include a computing device 118 having a processor 120 and a memory 122. The client device 104 includes a rendering device 124 to provide a tangible output of data which is communicatively coupled to the computing device 118 over a local connection. Like the source device 102, although the client device 104 is shown as including a separate computing device 118 and a separate rendering device 124, the client device 104 may be configured in a variety of ways. For instance, the client device 104 may be configured as a network-ready rendering device that does not include the full resources of the computing device 118. For example, the client device 104 may be configured as a low resource device that has limited processing and memory capabilities, such as a limited amount of RAM, no disk drive, and limited processing capabilities.

[0043] A parent distributed media session 126 (parent DMS) is included to allow the system 100 to stream data from the source device 102 over the network 106 to the client device 104. The parent DMS 126 is a software component that may be executed by the source device 102, the client device 104, and/or a third party device 2002 seen in FIG. 20. Through execution of the parent DMS 126, a distributed software infrastructure may be built to stream data over the network 106. The parent DMS 126, when executed, may discover the capabilities of both the source and client devices 102, 104. From these discovered capabilities, the parent DMS 126 may address (a or the?) distribution of

software components across both the client device 104 and the source device 102. Further discussion of software component distribution is found in the section titled "Resolving a Distributed Topology", below. To aid the following discussion, further references to execution of the parent DMS 126 or other software component to perform an action will be made as if the parent DMS 126 or other software component performs the action.

[0044] FIG. 2 is an illustration of a computing environment 200 utilizing the distributed media session 126 to stream data. The parent DMS 126 may be included as a part of an operating system 202 that is executed on the source device 102 or the client device 104 shown in FIG. 1, as well as part of an operating system for a third party device 2002 seen in FIG. 20. The operating system 202 provides for organization and control of software and hardware, such as source peripheral devices 204(1)-204(N) and rendering devices 206(1)-206(M).

[0045] The parent DMS 126 may build on top of a local software infrastructure which includes local software components 210 used for input and/or output of streams of data. Local software components 210 may include device drivers 212, players 214 for output of a stream of data such as media players, and other 216 software components. The parent DMS 126 may manage these local software components 210 and stream data both to and from these local software components 210 over the network 106 shown in FIG. 1. Thus, the parent DMS 126 may complement the local software components 210 without replacing them.

[0046] A media engine 218 may be included to provide an interface between the application 208 and the parent DMS 126. The media engine 218 is utilized to present

capabilities of the parent DMS 126 to the application 208. In other words, the media engine 218 defines an application programming interface (API) so that the application 208 may interact with the parent DMS 126 and vice versa. Additionally, the parent DMS 126 may utilize the media engine 218 to provide an interface for interaction with local software components 210. The media engine 218 may also be utilized to control flow of data between software components. The media engine 218, for instance, may route a stream of data between local software components 210, initiate execution of the local software components 210, and so on.

[0047] Remote Access and Federation

Additional complications may be encountered in the building and control of a distributed software infrastructure, as opposed to the building and control of local software components 210, in that the software components are executed on two or more devices in the distributed software infrastructure. Additionally, a variety of software components may be utilized in the distributed software infrastructure to stream data, which further complicates the building and control of the distributed software infrastructure. For example, the variety of software components may include local software components 210, media engines 218, transmitters, receivers, and the like, the use of which will be described in greater detail in the section titled "Resolving a Distributed Topology", below.

[0048] The parent DMS 126 provides a federated, i.e. centralized, approach for building a distributed software infrastructure that is suitable to stream data between two or more devices. This approach centralizes the planning process and may also centralize control of the software components. The parent DMS 126 may take into account the capabilities

of both the source and client devices 102, 104 when building the distributed software infrastructure. This allows software components of both the source and client devices 102, 104 to be built and controlled in unison by the parent DMS 126. By using the federated approach, the parent DMS 126 may provide a distributed software infrastructure which is optimized to the capabilities of the source and client devices, as will be further discussed in relation to FIG. 25 below.

[0049] Even though a federated approach may be taken to building the distributed topology, elements of building and control may be provided in separate portions of the distributed topology across multiple devices. For example, building and control may be delegated to devices having host affinities for the software components which are utilized to stream data from the source device 102 to the client device 104. This model is supported by the parent DMS 126 through use of parent-child relationships. Rather than attempting to directly control all of the software components of the distributed software infrastructure directly, the parent DMS 126 may create, e.g. instantiate, a child DMS 302 to which it delegates control, as shown in FIG. 3. The resulting structure can be a simple, single parent-child relationship as shown between the parent DMS 126 and the child DMS 302, or a more complex tree-like structure comprising several parent-child relationships as shown in FIG. 3.

[0050] FIG. 4 is an illustration of an exemplary implementation which depicts the parent DMS 126 as providing remote access functionality using a federated approach. The parent DMS 126 acts as a parent to host and control the child DMS 302. The parent DMS 126 may provide control over software components it hosts to the application 208 through the media engine 218. In this way, the parent DMS 126 provides remote access

functionality of the streams of data to the application 208. For example, the parent DMS 126 hosts the child DMS 302, which includes a media engine 402 that is executed across a device boundary 404, i.e. is executed on a different device. Through the child DMS 302, the parent DMS 126 may control the media engine 402 across the device boundary 404. Thus, the child DMS 302 provides the functionality of the media engine 402 to the application 208 through the parent DMS 126, even though the media engine 402 is remote to the application 208.

[0051] The parent DMS 126 may also directly host a media engine 406. The parent DMS 126 has the ability to host several types of software components, which may include the media engine 406, child DMS 302, and local software components 210 shown in FIG. 2 which are used to interact with universal plug and play devices, legacy devices, and so on. By hosting multiple software components, the parent DMS 126 may aggregate control over the software components. Therefore, multiple software components may be controlled by the application 208 through interaction with the parent DMS 126.

[0052] The child DMS 302 may be configured in a variety of ways to provide different functionality. For example, the child DMS 302 may be created to address a situation in which the parent DMS 126 does not have direct access to a software component, e.g. the media engine 402, for which the child DMS 302 has access. Therefore, the child DMS 302 is hosted by the parent DMS 126 to provide access to the software component, e.g. the media engine 402. Additionally, the child DMS 302 may be created to exert control over a specified set of software components. For example, a separate child DMS may be provided for each type of data that is streamed, such as video data and audio data.

[0053] Ultimately, the parent DMS 126 may provide aggregated control of at least a portion of the distributed software infrastructure, as shown in FIG. 4. For example, the parent DMS 126 may control at least one software component located on each of the source and client devices 102, 104, respectively. The decision to produce the child DMS 302 is the result of resolving a distributed topology, which is discussed in the following section.

[0054] **Resolving a Distributed Topology**

FIG. 5 is an illustration of an exemplary procedure 500 in which the parent DMS 126 builds a distributed software infrastructure for streaming data from the source device 102 to the client device 104. At block 502, a requesting step is shown wherein the application 208 sends a request 504 to the parent DMS 126 to stream data. The request references the source and client devices 102, 104 shown in FIG. 1.

[0055] The parent DMS 126 uses the request 504 to resolve a distributed topology that references software components that are suitable to stream data from the source device 102 to the client device 104. To resolve the distributed topology, the parent DMS 126 first discovers capabilities of the source device 102 and the client device 104, respectively, to interact with streaming data. For example, the DMS 126 may discover capabilities of the client device 104 to render data and capabilities of the source device 102 to stream the data to be rendered.

[0056] The capabilities may be discovered in a variety of ways. In the illustrated implementation at block 506, a discovering capabilities step is shown wherein the parent DMS 126 may examine the source device 102 to find a source software component 508 that involves, i.e. has as a feature or consequence, the streaming of data from the source

device 102. Likewise, the parent DMS 126 may examine the client device 104 to find a client software component 510 that involves rendering data which is streamed to the client device 104. Examining the source and client devices 102, 104 may include finding software components which are exposed, e.g. made visible, by the source and client devices 102, 104 to the parent DMS 126.

[0057] In another implementation, the parent DMS 126 may discover capabilities of the source and client devices 102, 104 by querying a look-up table. The look-up table may return a result that specifies capabilities of the source and/or client devices 102, 104. The look-up table may reside as a part of the parent DMS 126 and/or elsewhere over the network 106 shown in FIG. 1, such as on the source device 102, client device 104, and/or an additional device (not shown) which is accessible over the network 106.

[0058] At block 512, a resolving a distributed topology step is shown wherein the parent DMS 126 derives a distributed topology 514 from the discovered capabilities of both the source and client devices 102, 104 found at block 506. The distributed topology 514 is a plan, e.g. a blueprint, which references software components that, when the software components are executed, stream data from the source device 102 to the client device 104. For example, the source software component 508 discovered by the parent DMS 126 in the source device 102 may be a driver for the source peripheral device 108 shown in FIG. 1. Additionally, the client software component 510 discovered in the client device 104 may be a media player which is used to provide data through a device driver for the rendering device 124 of the client device 104 shown in FIG. 1. Based on the discoveries the parent DMS 126 may derive an additional software component 516 that, when executed in conjunction with the source software component 508 and the client

software component 510, streams data from the source device 102 to the client device 104. The parent DMS 126 derives the distributed topology 514 so that it includes the additional software component 516 based on the discovered capabilities of the source and client software components 508, 510. In the illustrated example, the distributed topology 514 references the source and client software components 508, 510 along with the additional software component 516. At this point distributed topology 514 provides a plan for streaming data from the source device 102 to the client device 104, but does not provide the actual distributed software infrastructure to do so.

[0059] At block 518, a building a distributed software infrastructure step is shown wherein the parent DMS 126 supplies the additional software component 516, based on the distributed topology 514, to build a distributed software infrastructure 520. The distributed software infrastructure 520 includes the additional software component 516 and the source and client software components 508, 510. The source, client and additional software components 508, 510, 516 of the distributed software infrastructure 520 are executed to stream data from the source device 102 to the client device 104. The additional software component 516 may be supplied in a variety of ways. For example, the additional software component 516 may be stored and retrieved by the parent DMS 126, such as from a local memory, memory located over the network 106 of FIG. 1, and the like.

[0060] In the illustrated example of FIG. 5, the parent DMS 126 starts with the request 504 which includes a reference to the source and client devices 102, 104. The parent DMS 126 then resolves from the request 504 the distributed topology 514 of block 512, which may then be used to build the distributed software infrastructure 520 of block 518.

The steps described at blocks 502, 506, 512, 518 may be performed automatically by the parent DMS 126 without user intervention. The parent DMS 126 may resolve a variety of distributed topologies, such as a remote source distributed topology, a remote sink distributed topology and a third party distributed topology, each of which will be further discussed in the following sections.

[0061] Remote Source Distributed Topology

FIGS. 6, 7 and 8 are illustrations of an exemplary implementation showing resolution of a remote source distributed topology. The remote source distributed topology refers to a distributed topology in which the source device 102 of the streaming data is remote, i.e. across the device boundary 404, from the client device 104 which hosts the parent DMS 126. The parent DMS 126 is executed by the client device 104 to resolve the distributed topology. The application 208 is also executed on the client device 104. An implementation is shown in FIG. 7, where the source device 102 includes the camera 108a and a camera software component 602, and the client device 104 includes the rendering device 124 and a display software component 604.

[0062] FIG. 6 illustrates an exemplary implementation of resolving a distributed topology at block 512 of procedure 500 of FIG. 5. The parent DMS 126 first creates a distributed topology 606 which includes the camera software component 602 and the display software component 604. The camera software component 602 references capabilities of the camera 108a of FIG. 7, such as local software components included on the camera 108a which stream data, and a network address of the camera 108a. Likewise, the display software component 604 may be used to reference capabilities,

such as which local software components are included on the rendering device 124 of FIG. 7 to render data and a network address of the rendering device 124.

[0063] The distributed topology 606 shown in FIG. 6 is partial because all the software components are not referenced that will be used to provide streaming data. As previously stated, the distributed topology 606 is a plan for a distributed software infrastructure and, as such, does not include the actual software components which will be implemented. For example, the distributed topology 606 references software components that will be used by the parent DMS 126 to provide the distributed software infrastructure for streaming data from the camera 108a to the rendering device 124 of FIG. 7. These references may be used by the parent DMS 126 to supply software components referenced in the distributed topology 606 to build a distributed software infrastructure as discussed in relation to FIG. 5.

[0064] In the illustrated example of FIG. 7, the camera 108a and the rendering device 124 are not locally connected to each other, but are instead connected over the network 106 shown in FIG. 1. As was discussed in the section titled "Remote Access and Federation" and FIG. 3, federation may involve segmenting the distributed topology 606 by delegating control to the child DMS 302 that will control local software components that will be used to stream data in the distributed software infrastructure. For example, as shown in FIG. 7, the distributed topology 606 is segmented to include a child DMS 702 being created on the remote source, i.e. the source device 102, which is separated by the device boundary 404 from the client device 104.

[0065] The child DMS 702 includes a portion 704 of the distributed topology 606 which will also undergo a resolution step. For example, resolving the portion 704 may include

adding a reference to a transmitter 706 that is a software component that is controlled by the parent DMS 126 through the child DMS 702. Likewise, resolving the distributed topology 606 may also include adding a reference to a receiver 708 which is a software component that is directly controlled by the parent DMS 126. The transmitter 706 streams data across the device boundary 404. The receiver 708 receives the data which was streamed across the device boundary 404. Data may be streamed between the transmitter and receiver 706, 708 in a variety of ways. For example, data may be "pushed" to the receiver 708 by the transmitter 706, such as by streaming data to a network address. Data may also be "pulled" from the transmitter 706 by the receiver 708, such as by retrieving a stream of data from a network address.

[0066] FIG. 8 illustrates the distributed topology 606 of FIGS. 6-7 as fully resolved. In this instance, a media engine 802 is referenced in the portion 704 of the distributed topology 606 to control streaming of data from the camera software component 602 to the transmitter 706. Another media engine 804 is included in the distributed topology 606 to control streaming of data from the receiver 708 to the display software component 604. The arrows illustrated in FIG. 8 depict the streaming of the data.

[0067] FIG. 9 is an illustration of a hierarchy of control 900 of software components in the distributed topology 606 shown in FIG. 8. The distributed topology 606 as shown in FIG. 8 is resolved such that the parent DMS 126 has control over all the software components of the distributed topology. Therefore, the parent DMS 126 will have control over a distributed software infrastructure that is built from the distributed topology 606. For instance, the parent DMS 126 has control over the camera software component 602, the media engine 802 and the transmitter 706 through use of the child

DMS 702. In this instance, the child DMS 702 resides on the source device 102 as shown in FIG. 7. The parent DMS 126 also has control over the receiver 708, media engine 804 and display software component 604 which are included with the parent DMS 126 on the client device 104 as shown in FIG. 7. Thus, the parent DMS 126 will have control over a distributed software infrastructure that may be built from the distributed topology 606, and may provide this control to the application 208 which is interacting with the parent DMS 126 through the media engine 218.

[0068] Remote Sink Distributed Topology

FIG. 10 is an illustration of an exemplary implementation showing a resolved remote sink distributed topology 1000??. The remote sink distributed topology 1000? refers to a distributed topology in which the client device 104 is remote, i.e. across a device boundary 404, from the source device 102 which executes the parent DMS 126. The parent DMS 126 is executed by the source device 102 to resolve the distributed topology. The distributed topology for a remote sink scenario is effectively a mirror image of the remote source scenario as described in relation to FIGS. 6-9.

[0069] A distributed topology 1002 is resolved which references the camera software component 602 and display software component 604 of the camera 108a and rendering device 124 as described in the previous implementation shown in FIGS. 6-9. As before, the parent DMS 126 segments the distributed topology 1002 by creating a child DMS 1004 on the client device 104 which is separated from the source device 102 that is hosting the parent DMS 126 across the device boundary 404. The child DMS 1004 includes a portion 1006 of the distributed topology 1002. The distributed topology 1002

references a transmitter 1008 and a media engine 1010 for inclusion on the source device 102, and a receiver 1012 and a media engine 1014 for inclusion on the client device 104.

[0070] FIG. 11 is an illustration of a hierarchy of control 1100 of software components referenced in the remote sink distributed topology 1000 shown in FIG. 10. The parent DMS 126 has control over a distributed software infrastructure that may be formed from the distributed topology 1002 of FIG. 10. For instance, the parent DMS 126 has direct control over the camera software component 602, the media engine 1010 and the transmitter 1008. The parent DMS 126 also has control over the receiver 1012, the media engine 1014 and the display software component 604 which are included with the child DMS 702 on the client device 104 as shown in FIG. 10. Thus, the parent DMS 126 has control over the distributed software infrastructure that may be built from the distributed topology 1002. The application 208, interacting with the parent DMS 126 through the media engine 218, may then also control a distributed software infrastructure formed from the distributed topology 1002 of FIG. 10.

[0071] In both of the previous implementations of the remote source distributed topology and the remote sink distributed topology, the parent DMS 126 provides control over the entire distributed software infrastructure built from the distributed topologies. There may be instances, however, in which control over the entire distributed software infrastructure is not provided, as will be discussed in the following section.

[0072] Source and Sink Proxy Software Components for use in Distributed Topologies

FIG. 12 is an illustration of an exemplary implementation 1200 of a proxy 1202. The proxy 1202 is a software component through which the parent DMS 126 (not shown) may provide interaction to the application 208 with a software component 1204 that is

located on a different device, e.g. across the device boundary 404, from the application 208. The proxy 1202 is utilized to present the streaming data to the application 208 as if the data was provided locally through use of a local software component 210. Thus, the application 208 may receive streaming data from the camera 108a as though the camera 108a and its corresponding software component 1204 were executed on the client device 104.

[0073] FIG. 13 is an illustration of an additional exemplary implementation 1300 of the proxy 1202. The proxy 1202 may also be utilized to receive and stream data across the device boundary 404 to a software component 1302. For example, the proxy 1202 may stream data to the software component 1302 for display on the rendering device 124 as though the software component 1302 was a local software component 210 being executed on the source device 102.

[0074] The proxy 1202 may be utilized in instances in which control over all the software components of the distributed software infrastructure is not provided by the parent DMS 126 (not shown). Therefore, through use of the proxy 1202, a federated model is utilized in the resolving of the distributed topology by the parent DMS 126, but a federated model is not utilized in control as was the case in relation to FIGS. 6-11.

[0075] FIG. 14 is an illustration showing an exemplary implementation 1400 of a source proxy 1402. The source proxy 1402 includes the parent DMS 126 and a receiver 1404. The parent DMS 126 is used for control of remote software components that reside on the source device 102 shown in FIG. 12. The receiver 1404 receives data that is streamed by a transmitter (not shown) to the receiver 1206 as described previously.

[0076] FIG. 15 is an illustration showing an exemplary implementation 1500 of a sink proxy 1502. The sink proxy 1502 includes the parent DMS 126 which is used for control of the software component 1302 that resides on the client device 104 as shown in FIG. 13. The transmitter 1504 streams data to a receiver (not shown) as described previously. The sink proxy 1502 may be used to stream data from a software component without using the parent DMS 126 to control the software component which is acting as a source of the data.

[0077] FIG. 16 is an illustration of an exemplary implementation showing a remote source distributed topology 1600 that includes a source proxy 1602. The application 208 and media engine 218 are executed on the client device 104 as described in the remote source distributed topology shown in FIGS. 6-9. In this instance, however, the parent DMS 126 that receives the request from the application 208 to stream data will not be used for control of each of the software components of the source device 102 which involve the data to be streamed. In other words, the parent DMS 126 will not exert control over the entire distributed software infrastructure which is built from the distributed topology. Therefore, when resolving the distributed topology, the parent DMS 126 references the source proxy 1602 so that the data is streamed to the software component that is not controlled by the parent DMS 126.

[0078] The parent DMS 126 provides control over software components which reside on the source device 102 to stream data from the source device 102 to the client device 104 which hosts the parent DMS 126. For instance, the parent DMS 126 may control a source software component 1604 and a transmitter 1606 through use of a media engine 1608. The transmitter 1606 streams data to the receiver 1610 as previously described.

The parent DMS 126, however, does not have control over the destination of the data, which in this instance will be referred to as a sink 1612. The sink 1612 is a software component that is illustrated to indicate a destination for the data that is not controlled by the parent DMS 126. Therefore, the parent DMS 126 simply streams the data to the sink 1612 without controlling it. The application 208 may interact with the source proxy 1602 as though it is a local software components 1614, even though the parent DMS 126 controls software components that are remote to the application 208, e.g. located on the source device 102.

[0079] FIG. 17 is an illustration which depicts a hierarchy 1700 of control in the remote source distributed topology 1600 as shown in FIG. 16. In the remote source distributed topology 1600, the parent DMS 126 provides control over the media engine 1608, source software component 1604, transmitter 1606 and receiver 1610 as previously described. Because the parent DMS 126 does not exert control over the sink 1612, the parent DMS 126 recognizes the sink 1612 as an outlet where the data is to be streamed, but is not controlled.

[0080] FIG. 18 is an illustration showing an exemplary implementation of a remote sink distributed topology 1800 that includes a sink proxy 1802. The remote sink distributed topology 1800 is similar to the remote source distributed topology 1600. In this implementation, the application 208 and media engine 218 are executed on the source device 102 as described in relation to the remote sink distributed topology shown in FIGS. 10-11.

[0081] The parent DMS 126 that receives the request from the application 208 to stream data will not be used in a distributed software infrastructure for control of all of the

software components of the client device 104 which involve the streaming of data. Therefore, the sink proxy 1802 is referenced by the parent DMS 126 in response to the request so that the data to be streamed may be received utilizing a software component over which the parent DMS 126 does not have control.

[0082] The parent DMS 126 provides control over software components that reside on the client device 104 that receives data streamed from the source device 102. For instance, the parent DMS 126 may control a receiver 1804 and a client software component 1806 through use of a media engine 1808. A transmitter 1810 is used to stream data to the receiver 1804 as previously described. The parent DMS 126, however, does not have control over all the software components which stream data, which in this instance will be referred to as an origin 1812. Therefore, the parent DMS 126 simply receives the data from the origin 1812 without controlling it. The application 208 may recognize the sink proxy 1802 as a local software component 1814.

[0083] FIG. 19 is an illustration which depicts a hierarchy 1900 of control in the remote sink distributed topology 1800 shown in FIG. 18. The parent DMS 126 provides control over the media engine 1808, client software component 1806, receiver 1804 and transmitter 1810. Because the parent DMS 126 does not exert control over the origin 1812, the parent DMS 126 recognizes the origin 1812 as an inlet from which data is received, but over which the parent DMS 126 does not have control.

[0084] Third Party Distributed Topology

FIG. 20 is an illustration of an exemplary implementation in which a system 2000 is shown that includes the third party device 2002 which is communicatively coupled to the source and client devices 102,. 104 over the network 106. In the previous

implementations, distributed topologies were described in which resolution and control was provided by either the source device 102 or the client device 104. The third party device 2002 may also be utilized to execute the parent DMS 126 through use of the federated approach provided by the parent DMS 126. Therefore, the third party device 2002 may be utilized to resolve a distributed topology to stream data from the source device 102 to the client device 104.

[0085] FIG. 21 is an illustration of an exemplary implementation in which a third party distributed topology 2100 is shown. The third party device 2002 executes the parent DMS 126 which receives a request from the application 208 through the media engine 218 to stream data from the source device 102 to the client device 104. The parent DMS 126 creates a first child DMS 2102 to resolve a portion of the distributed topology which includes the source device 102. The first child DMS 2102 also provides control over software components included on the source device 102. The parent DMS 126 also creates a second child DMS 2104 to resolve a portion of the distributed topology of the client device 104. The second child DMS 2104 provides control over software components included on the client device 104.

[0086] The first child DMS 2102 references a source software component 2106 and a transmitter 2108 which are controlled by a media engine 2110. The second child DMS 2104 references a receiver 2112 and a client software component 2114 which are controlled by a media engine 2116. FIG. 22 is an illustration of a hierarchy 2200 of control in the third party distributed topology 2100 depicted in FIG. 21. The hierarchy 2200 demonstrates that the parent DMS 126 provides control over the first child DMS 2102 and second child DMS 2104. The first child DMS 2102 provides control over the

source software component 2106, the transmitter 2108 and the media engine 2100. The second child DMS 2104 provides control over the receiver 2112, the client software component 2114, and the media engine 2116.

[0087] The third party control distributed topology 2100 may be configured in a variety of ways. In the example illustrated in FIG. 21, the parent DMS 126 provides control over the software components which are located on the source and client devices 102, 104. The third party device 2002 of FIG. 20 may also execute the parent DMS 126 to resolve a distributed topology in which control is provided by either the source device 102 or the client device 104. Also, it should be noted that in the illustrated implementation shown in FIG. 21, the data is streamed from the source device 102 to the client device 104 without being streamed through the third party device 2002. In this way, the third party device 2002 may resolve a distributed topology that streams data without actually streaming the data itself. In additional implementations, data is streamed from the source device 102 through the third party device 2002 to the client device 104.

[0088] Distributed Topology having Multiple Streams

Although the previous implementations were described in relation to a single stream of data to aid the discussion, multiple streams of data may be streamed by the parent DMS 126. For example, multiple source devices may be utilized, in which each of the source devices stream data to a single client device. Additionally, a single source device may stream multiple streams of data to one or more client devices. Likewise, multiple client devices may receive a single stream of data from a source device. Even though a single example of use of multiple streams of streaming data will be discussed,

each of the distributed topologies and distributed software infrastructures previously discussed may support a plurality of streams of data.

[0089] FIG. 23 is an illustration of an exemplary implementation of a system 2300 in which a plurality of source devices stream data to the client device 104. The system 2300 may include a first source device 2302 which is configured as a network-ready camera, e.g. it contains functionality which enables it to be communicatively coupled to the network 106. The system 2300 also includes a second source device 2304 which is configured as a network-ready audio input device, such as a network-ready microphone. The first source device 2302 streams video data and the second source device 2304 streams audio data for use by the client device 104. The first and second source devices 2302, 2304 may be configured in a variety of ways, such as universal plug-and-play (UPnP) devices.

[0090] FIG. 24 is an illustration of an exemplary implementation showing a distributed topology for use in the system 2300 of FIG. 23. The client device 104 includes the application 208 which requests the streaming of data from both the first and second source devices 2302, 2304 for rendering at the client device 104. The parent DMS 126 receives the request through the media engine 218 and resolves a distributed topology for streaming the audio and video data.

[0091] The parent DMS 126 creates a first child DMS 2402 to resolve a portion of the distributed topology of the first source device 2302. The first child DMS 2402 also provides control of software components included on the first source device 2302. Additionally, the parent DMS 126 creates a second child DMS 2404 to resolve a portion

of the distributed topology of the second source device 2302. The second child DMS 2404 also controls of software components included on the second source device 2302.

[0092] The first child DMS 2402 references a camera software component 2406 and a transmitter 2408 which are controlled by a media engine 2410. The second child DMS 2404 references an audio software component 2412 and a transmitter 2414 which are controlled by a media engine 2416. The parent DMS 126 may control the first and second child DMS 2402, 2404 to coordinate the streaming of the audio and video data to the client device 104.

[0093] Resolving Distributed Topologies in an Optimized Manner

FIG. 25 is an illustration of an exemplary implementation of a system 2500 in which the parent DMS 126 resolves the distributed topology in an optimized manner based on the capabilities of the source and client devices 102, 104 to interact with streaming data. To render data locally, the source device 102 may produce a raw stream of data 2502, i.e. data that has not been processed or subjected to analysis, from an included source peripheral device 108. The raw data 2502 is compressed to form compressed data 2504 so that the data may be transmitted and stored in an efficient manner. To interact with the compressed data, the source device 102 decompressed the compressed data 2504 to obtain decompressed data 2506. The decompressed data 2506 then went through a rendering step to form rendered data 2508, i.e. data that is suitable for output in a tangible form for interaction with by a user (such as displayed, played by an audio device, etc.).

[0094] The parent DMS 126 may provide an optimized distributed software topology by discovering the capabilities of both the source device 102 and the client device 104. For

example, as illustrated in FIG. 25, the parent DMS 126 may resolve a distributed topology in which the compressed data 2504 obtained from the raw data 2502 is streamed to the client device 104. The client device 104 may then decompress the compressed data 2504 to form decompressed data 2510. The decompressed data 2510 is then rendered to form rendered data 2512 for use by the rendering device 124.

[0095] Optimization of the distributed topology may be used to provide a distributed software infrastructure that distributes the resources used to stream data. For example, rather than provide the source device 102 with sufficient hardware and software resources to decompress and render, this functionality may be provided by the client device 104. Additionally, the compressed data 2504 may be compressed in such a way as to be encrypted. By providing for the encryption of compressed data 2504, the data remains encoded, and therefore, may be safely streamed over the network 106.

[0096] **Exemplary Procedure**

FIG. 26 is a flow chart depicting an exemplary procedure 2600 in which a distributed software infrastructure of software components is built which streams data from the source device 102 to the client device 104. At block 2602, a request is sent by the application 208. The request may include an instruction to communicate streaming data from the source device 102 to the client device 104. The request may be thought of as a partial topology in that it does not include a description of the software components used to build the distributed software infrastructure to stream the data.

[0097] At block 2604, the parent DMS 126 receives the request which includes the partial topology of the source device 102 and the client device 104. At block 2606, the parent DMS 126 resolves a distributed topology. The distributed topology is a plan that

references software components suitable to stream data from the source device 102 over a network to the client device 104.

[0098] At block 2608, the parent DMS 126 first begins to resolve the distributed topology by discovering capabilities to interact with streaming data of both the source and client devices 102, 104. Capabilities may include software components which are used to stream data that is to be rendered and to render a stream of data by the source and client devices 102, 104, respectively. These capabilities may be discovered in a variety of ways. For example, the parent DMS 126 may examine the source and client devices 102, 104 to determine which software components are included that involve streaming data. The parent DMS 126 may also query a look-up table which includes descriptions of the capabilities of the source and client devices 102, 104.

[0099] At block 2610, the parent DMS 126 derives a distributed topology that references a plurality of software components which are suitable to fulfill the request. The distributed topology may reference software components which are already included on the source and client devices 102, 104, as well as additional software components which are to be included to stream data from the source device 102 to the client device 104.

[00100] At block 2612, the parent DMS 126 builds a distributed software infrastructure from the distributed topology which includes the software components referenced by the distributed topology. For instance, the parent DMS 126 may supply an additional software component to build a distributed software infrastructure based on the resolved distributed topology. Thus, the parent DMS 126 uses the distributed topology as a plan to supply software components which are referenced by the distributed topology. At block 2614, the source device 102 streams data to the client device 104 over the network

using the distributed software infrastructure. At block 2616, the client device 104 renders the data.

[00101] **Exemplary Operating Environment**

The various components and functionality described herein are implemented with a number of individual computers. FIG. 27 shows components of a typical example of a computer environment 2700, including a computer, referred by to reference numeral 2702. In various implementations, the source device 102, the client device 104, and/or the third party device 2002 may be configured as the computer 2702. The components shown in FIG. 27 are only examples, and are not intended to suggest any limitation as to the scope of the functionality of the invention; the invention is not necessarily dependent on the features shown in FIG. 27.

[00102] Generally, various different general purpose or special purpose computing system configurations can be used. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, network-ready devices, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

[00103] The functionality of the computers is embodied in many cases by computer-executable instructions, such as software components, that are executed by the computers. Generally, software components include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular

abstract data types. Tasks might also be performed by remote processing devices that are linked through a communications network. In a distributed computing environment, software components may be located in both local and remote computer storage media.

[00104] The instructions and/or software components are stored at different times in the various computer-readable media that are either part of the computer or that can be read by the computer. Programs are typically distributed, for example, on floppy disks, CD-ROMs, DVD, or some form of communication media such as a modulated signal. From there, they are installed or loaded into the secondary memory of a computer. At execution, they are loaded at least partially into the computer's primary electronic memory.

[00105] For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the data processor(s) of the computer.

[00106] With reference to FIG. 27, the components of computer 2702 may include, but are not limited to, a processing unit 2704, a system memory 2706, and a system bus 2708 that couples various system components including the system memory to the processing unit 2704. The system bus 2708 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA)

bus, Enhanced ISA (EISAA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as the Mezzanine bus.

[00107] Computer 2702 typically includes a variety of computer-readable media. Computer-readable media can be any available media that can be accessed by computer 2702 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. "Computer storage media" includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computer 2702. Communication media typically embodies computer-readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more if its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

[00108] The system memory 2706 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 2710 and random access memory (RAM) 2712. A basic input/output system 2714 (BIOS), containing the basic routines that help to transfer information between elements within computer 2702, such as during start-up, is typically stored in ROM 2710. RAM 2712 typically contains data and/or software components that are immediately accessible to and/or presently being operated on by processing unit 2704. By way of example, and not limitation, FIG. 27 illustrates operating system 2716, application programs 2718, software components 2720, and program data 2722.

[00109] The computer 2702 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. 27 illustrates a hard disk drive 2724 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 2726 that reads from or writes to a removable, nonvolatile magnetic disk 2728, and an optical disk drive 2730 that reads from or writes to a removable, nonvolatile optical disk 2732 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 2724 is typically connected to the system bus 2708 through a non-removable memory interface such as data media interface 2734, and magnetic disk drive 2726 and optical disk drive 2730 are typically connected to the system bus 2708 by a removable memory interface.

[00110] The drives and their associated computer storage media discussed above and illustrated in FIG. 27 provide storage of computer-readable instructions, data structures, software components, and other data for computer 2702. In FIG. 27, for example, hard disk drive 2724 is illustrated as storing operating system 2716', application programs 2718', software components 2720', and program data 2722'. Note that these components can either be the same as or different from operating system 2716, application programs 2718, software components 2720, and program data 2722. Operating system 2716', application programs 2718', software components 2720', and program data 2722'are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 2702 through input devices such as a keyboard 2736, and pointing device (not shown), commonly referred to as a mouse, trackball, or touch pad. Other input devices may include source peripheral devices (such as a microphone 2738 or camera 2740 which provide streaming data), joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 2702 through an input/output (I/O) interface 2742 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port, or a universal serial bus (USB). A monitor 2744 or other type of display device is also connected to the system bus 2708 via an interface, such as a video adapter 2746. In addition to the monitor 2744, computers may also include other peripheral rendering devices (e.g., speakers) and one or more printers 2748, which may be connected through the I/O interface 2742.

[00111] The computer may operate in a networked environment using logical connections to one or more remote computers, such as a remote device 2750. The remote device 2750

may be a personal computer, a network-ready device, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 2702. The logical connections depicted in FIG. 27 include a local area network (LAN) 2752 and a wide area network (WAN) 2754. Although the WAN 2754 shown in FIG. 27 is the Internet, the WAN 2754 may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the like.

[00112] When used in a LAN networking environment, the computer 2702 is connected to the LAN 2752 through a network interface or adapter 2756. When used in a WAN networking environment, the computer 2702 typically includes a modem 2758 or other means for establishing communications over the Internet 2754. The modem 2758, which may be internal or external, may be connected to the system bus 2708 via the I/O interface 2742, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 2702, or portions thereof, may be stored in the remote device 2750. By way of example, and not limitation, FIG. 27 illustrates remote software components 2760 as residing on remote device 2750. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

[00113] **Conclusion**

Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described.

Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.